

# Multiscale Computations using Finite Elements

**Jendrik-Alexander Tröger**, Stefan Hartmann

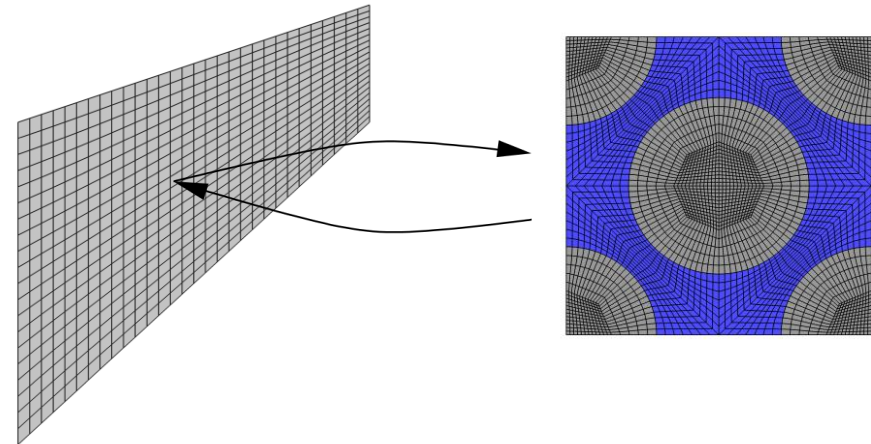
In collaboration with: H. Eivazi, S. Wittek, A. Rausch

Division of Solid Mechanics

Institute of Applied Mechanics

Clausthal University of Technology

[jendrik-alexander.troeger@tu-clausthal.de](mailto:jendrik-alexander.troeger@tu-clausthal.de)

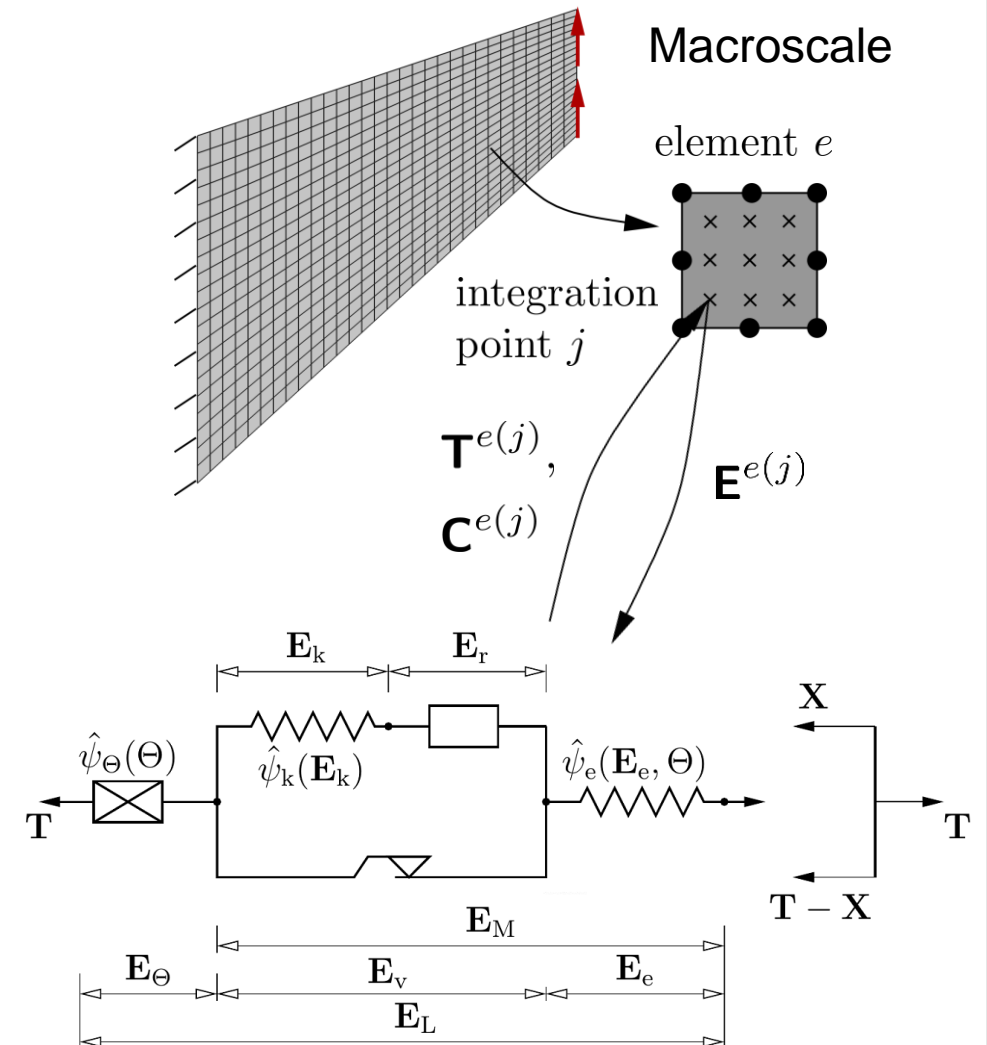


## Multiscale Computations using Finite Elements

- Consideration of heterogeneous microstructures in numerical simulations →  $FE^2$
- Numerical homogenization of microscale quantities
- Large number of microscale model evaluations within embedded structure  
→ High computational expenses

### Outline:

- Algorithmic structure of  $FE^2$
- Data-based deep neural network surrogates
- Application examples



## Multiscale Computations using Finite Elements

- Local balance of linear momentum

$$\operatorname{div} \mathbf{T}(\vec{x}, t) + \rho(\vec{x}) \vec{k} = \vec{0}$$

- Spatial discretization – Macroscale

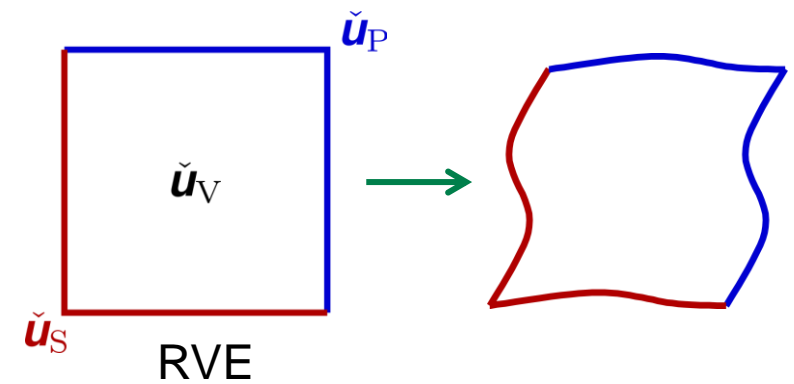
$$\mathbf{g}(t, \boldsymbol{\lambda}(t), \mathbf{T}(t)) = \mathbf{0} \longrightarrow \mathbf{G}(t, \mathbf{u}, \check{\mathbf{u}}, \check{\mathbf{q}}) = \mathbf{0}$$

- Spatial and temporal discretization – Microscale

$$\check{\mathbf{g}}(\check{\mathbf{u}}, \check{\boldsymbol{\lambda}}, \check{\mathbf{q}}^{e(j)}(t)) = \mathbf{0} \longrightarrow \check{\mathbf{G}}(t, \mathbf{u}, \check{\mathbf{u}}, \check{\mathbf{q}}) = \mathbf{0}$$

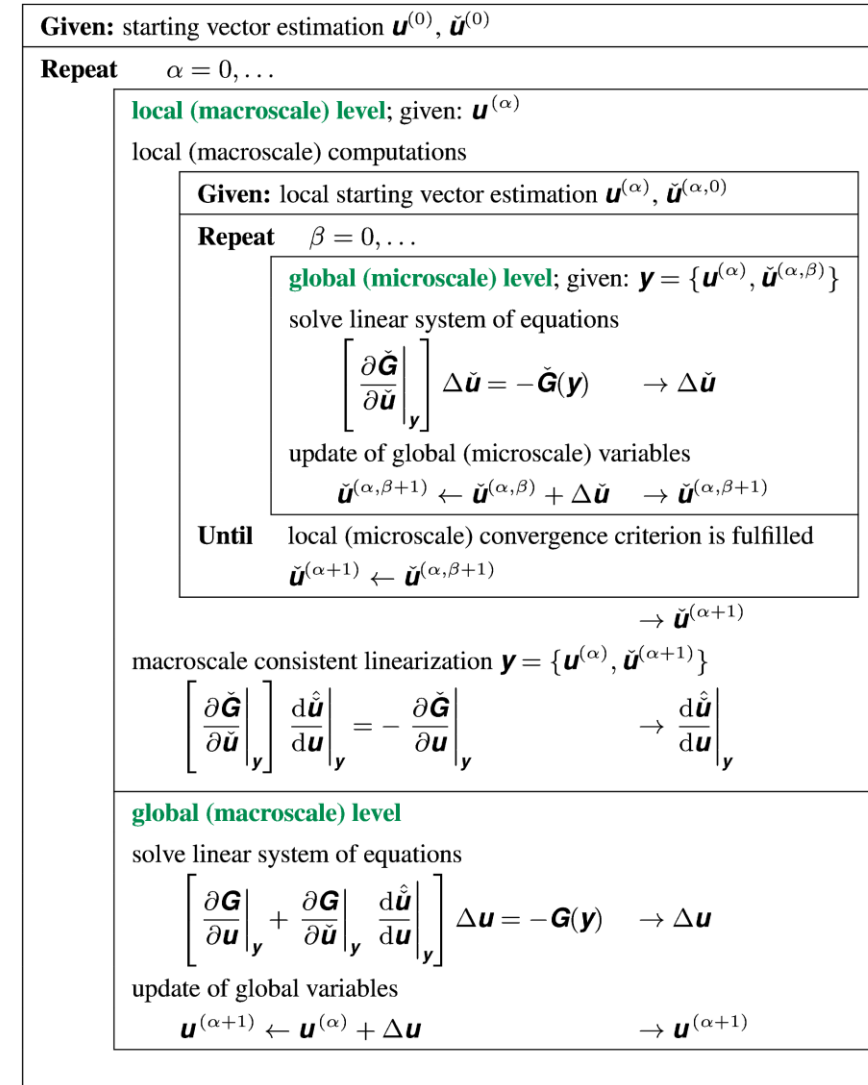
$$\dot{\check{\mathbf{q}}}^{e(j)}(t) - \check{\mathbf{r}}(\check{\mathbf{u}}_a, \check{\mathbf{q}}) = \mathbf{0} \longrightarrow \check{\mathbf{L}}(t, \mathbf{u}, \check{\mathbf{u}}, \check{\mathbf{q}}) = \mathbf{0}$$

- Periodic displacement boundary conditions on microscale enforced with constraints
- Computation of homogenized quantities using numerical homogenization  $\longrightarrow$  stress  $\mathbf{T}^{e(j)}$ , consistent tangent  $\mathbf{C}^{e(j)}$

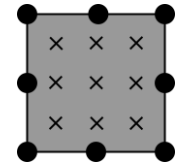


# Multilevel-Newton Algorithm – Purely Elastic FE<sup>2</sup>

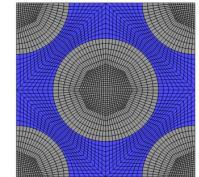
- Two-level Newton algorithm for purely elastic problems
  - Unknown quantities
    - Macroscale displacements  $\mathbf{u}$
    - Microscale displacements  $\tilde{\mathbf{u}}$
  - Iterative solution of systems of nonlinear equations
- Computational costs especially for local macroscale computations



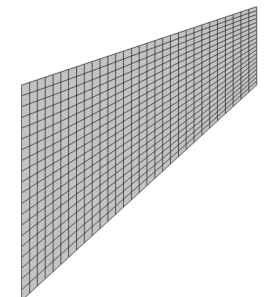
local level



RVE

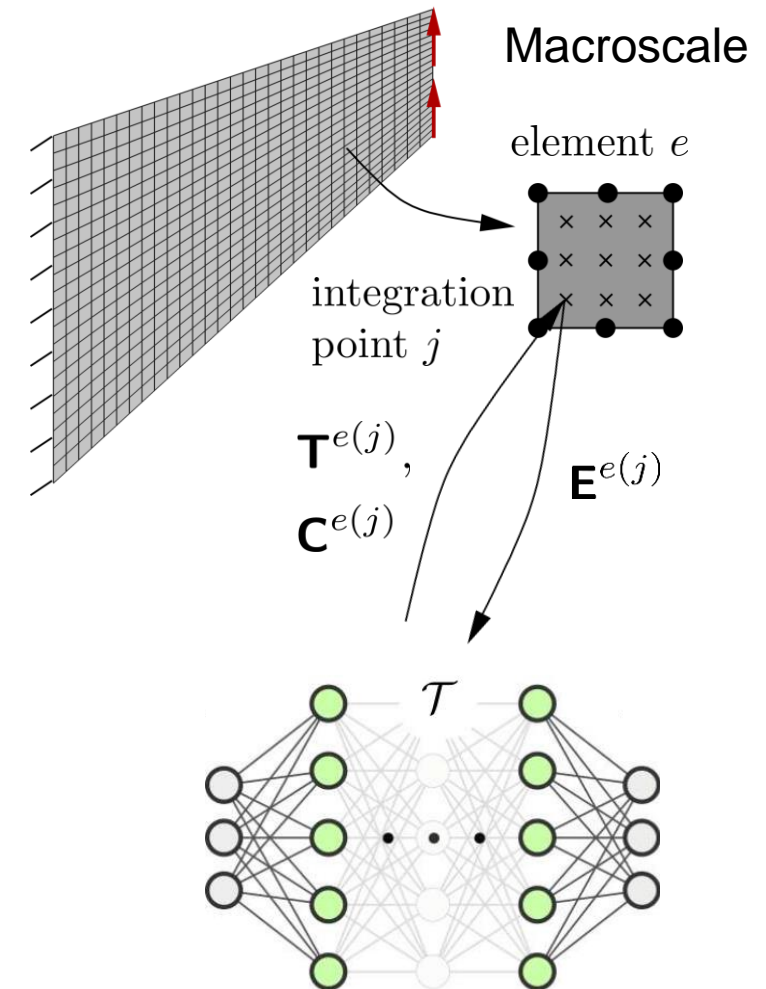
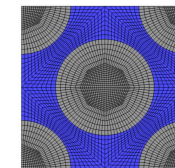


global level



## Multiscale Computations using Surrogate Models

- High computational expenses for full FE<sup>2</sup>
  - Deep neural networks (DNNs) as surrogate models for fast evaluation of mechanical microscale response
- As a first attempt, purely data-based DNN surrogate model
  1. Generation of training data using RVE computations and homogenization for various  $\mathbf{E}^{e(j)}$
  2. Training and validation of DNN surrogate model
  3. Efficient implementation into finite element code



## DNN-FE<sup>2</sup> – Newton Algorithm

- Application of DNN surrogate models to replace the local macroscale evaluations by function evaluations

$$\mathbf{T}^{e(j)} \approx \mathcal{T}(\mathbf{E}^{e(j)}(\mathbf{u}^{(\alpha)}); \theta_{\mathcal{T}})$$

$$\mathbf{C}^{e(j)} \approx \mathcal{C}(\mathbf{E}^{e(j)}(\mathbf{u}^{(\alpha)}); \theta_{\mathcal{C}})$$

- Integration point contributions

- Global tangential stiffness matrix

$$\mathbf{K} = \sum_{e=1}^{n_e} \mathbf{z}^{eT} \left[ \sum_{j=1}^{n_G^e} w_j \mathbf{B}^{e(j)T} \mathbf{C}^{e(j)} \mathbf{B}^{e(j)} \det \mathbf{J}^{e(j)} \right] \mathbf{z}^e$$

- Right hand side of linear system

$$\mathbf{G}(t_{n+1}, \mathbf{u}) = \sum_{e=1}^{n_e} \mathbf{z}^{eT} \left( \sum_{j=1}^{n_G^e} w_j \mathbf{B}^{e(j)T} \mathbf{T}^{e(j)} \det \mathbf{J}^{e(j)} \right) - \bar{\mathbf{p}}(t_{n+1})$$

**Given:** starting vector estimation  $\mathbf{u}^{(0)}$ ; surrogate parameters  $\theta_{\mathcal{T}}$  and  $\theta_{\mathcal{C}}$

**Repeat**  $\alpha = 0, \dots$

**local (macroscale) level;** given:  $\mathbf{u}^{(\alpha)}$

evaluate DNN surrogates for macroscale integration point  $j$  of element  $e$

$$\mathbf{T}^{e(j)} \approx \mathcal{T}(\mathbf{E}^{e(j)}(\mathbf{u}^{(\alpha)}); \theta_{\mathcal{T}})$$

$$\mathbf{C}^{e(j)} \approx \mathcal{C}(\mathbf{E}^{e(j)}(\mathbf{u}^{(\alpha)}); \theta_{\mathcal{C}})$$

**global (macroscale) level**

solve linear system of equations

$$\mathbf{K}^{(\alpha)} \Delta \mathbf{u} = -\mathbf{G}(\mathbf{u}^{(\alpha)}) \rightsquigarrow \Delta \mathbf{u}$$

update of global variables

$$\mathbf{u}^{(\alpha+1)} \leftarrow \mathbf{u}^{(\alpha)} + \Delta \mathbf{u} \rightsquigarrow \mathbf{u}^{(\alpha+1)}$$

**Until** global (macroscale) convergence criterion is fulfilled

## Development of DNN Surrogate Models

- Two-dimensional setup  $\rightarrow$  Input  $\bar{\mathbf{E}} \in \mathbb{R}^3$ , Output  $\bar{\mathbf{T}} \in \mathbb{R}^3, \bar{\mathbf{C}} \in \mathbb{R}^9$

- Model architectures (developed with TensorFlow)

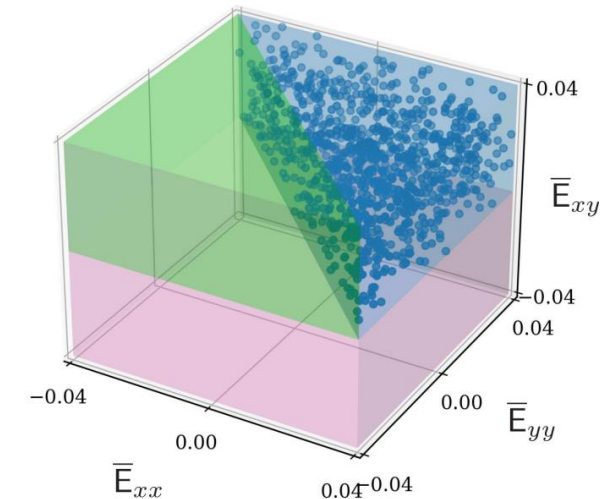
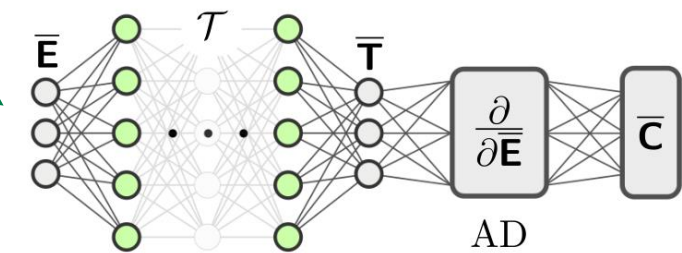
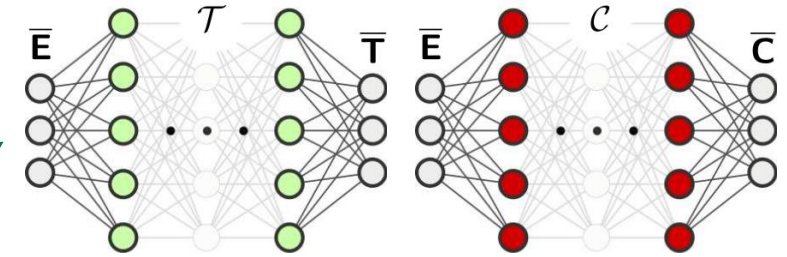
- Two separate DNN (NN-2)  $\bar{\mathbf{T}} = \mathcal{T}(\bar{\mathbf{E}}; \theta_{\mathcal{T}}), \quad \bar{\mathbf{C}} = \mathcal{C}(\bar{\mathbf{E}}; \theta_{\mathcal{C}})$

- One DNN + AD (NN-AD)  $\bar{\mathbf{T}} = \mathcal{T}(\bar{\mathbf{E}}; \theta_{\mathcal{T}}), \quad \bar{\mathbf{C}} = \mathcal{T}'(\bar{\mathbf{E}}; \theta_{\mathcal{T}})$

$\rightarrow$  Sobolev training to improve quality of prediction

- Settings:

- 8 hidden layers with 128 neurons
- *Swish* activation function
- Latin hypercube sampling for input space (min./max. strain  $\pm 0.04$ ) employing symmetry relations





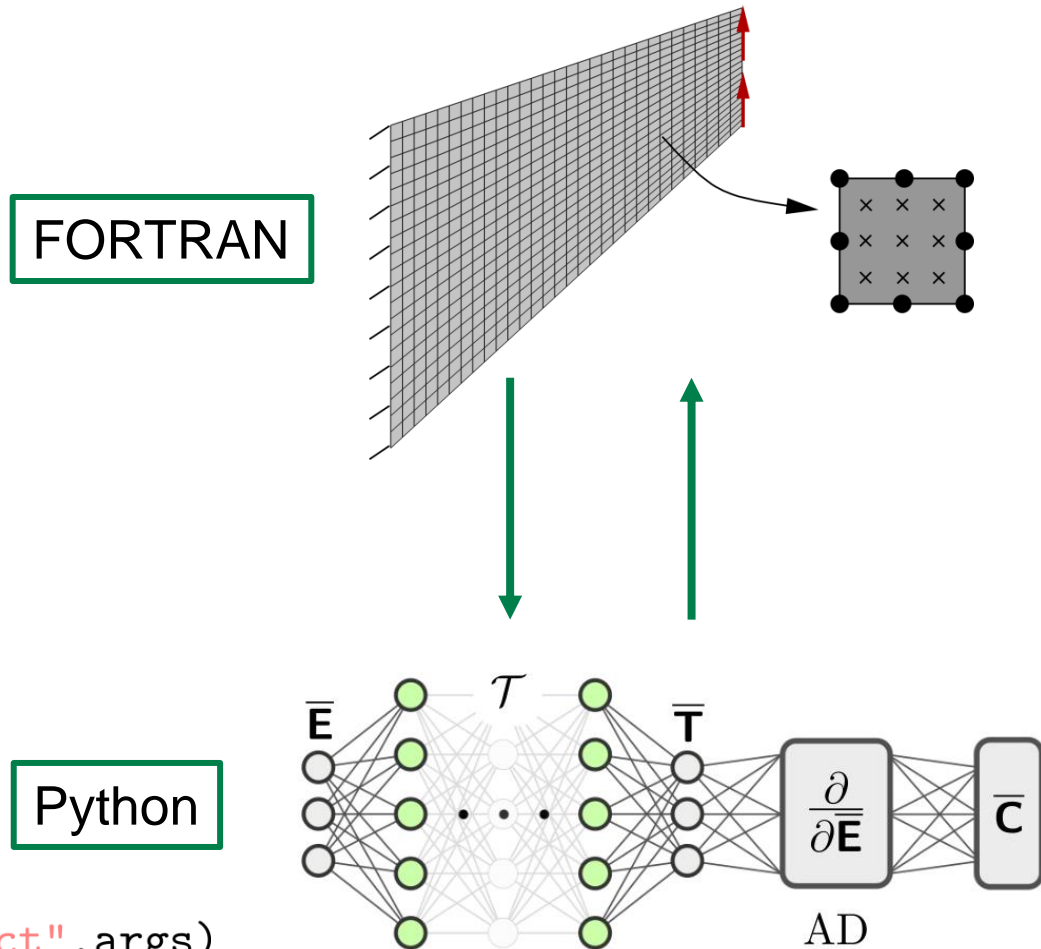
## Implementation – FORPy

- Efficient communication is required between FORTRAN finite element code and DNN implementation in Python

→ **FORPy library** (Rabel et al., 2020)

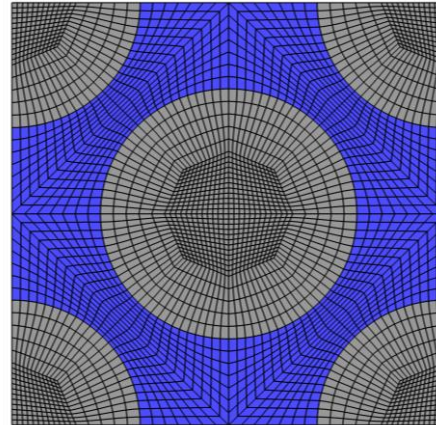
- Allows calling Python functions from FORTRAN
- Arguments are transferred via memory
- Suitability for parallelized code
- Example:

```
USE forpy_mod
...
ierr = forpy_initialize()
ierr = call_py(returnVal,pyModule,"predict",args)
CALL forpy_finalize
```





# APPLICATION EXAMPLES

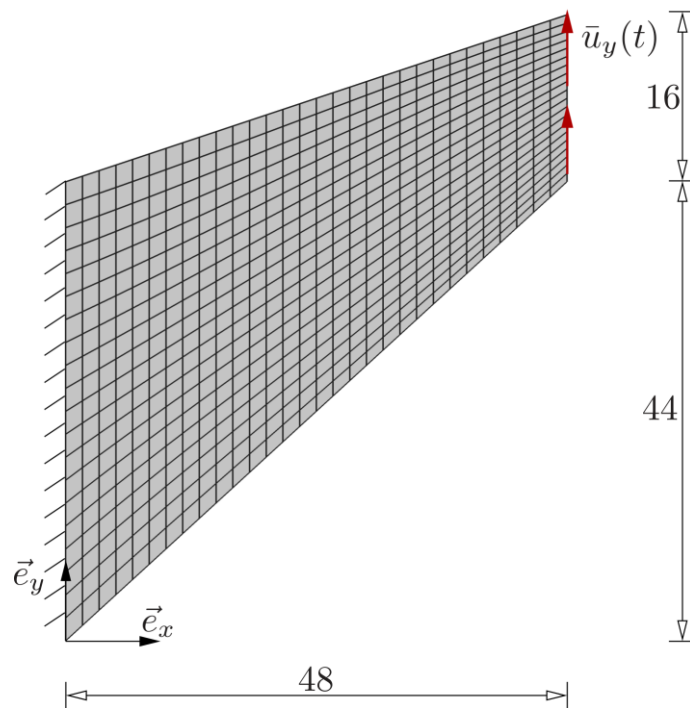


## Cook's Membrane – Setup

### ■ Macroscale – Cook's membrane

- Prescribed displacement

$$\bar{u}_y(t) = 2 \text{ mm s}^{-1} t$$



### ■ Microscale – RVE

- Non-linear elasticity

$$\mathbf{T} = K_m (\text{tr } \mathbf{E}) \mathbf{I} + G_m (\mathbf{E}^D) \mathbf{E}^D$$

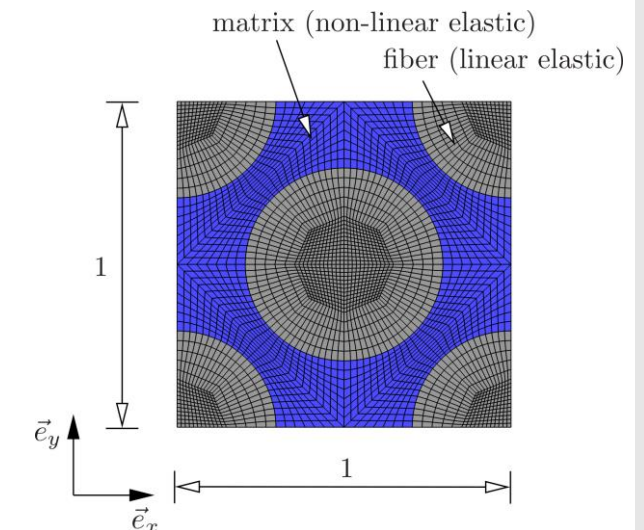
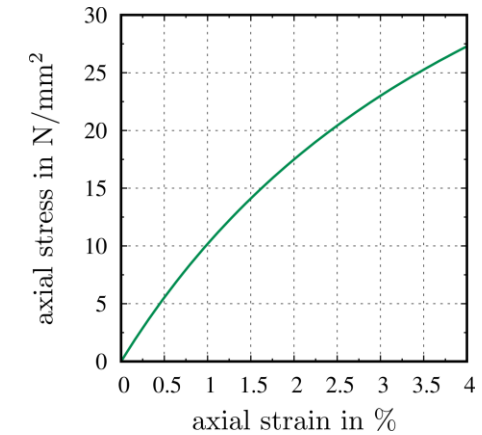
$$G_m (\mathbf{E}^D) = \frac{\alpha_1}{\alpha_2 + \|\mathbf{E}^D\|_2}$$

$K_m$ $\text{N mm}^{-2}$	$\alpha_1$ $\text{N mm}^{-2}$	$\alpha_2$ -
$4.78 \times 10^3$	$5.0 \times 10^1$	$6.0 \times 10^{-2}$

- Linear elasticity

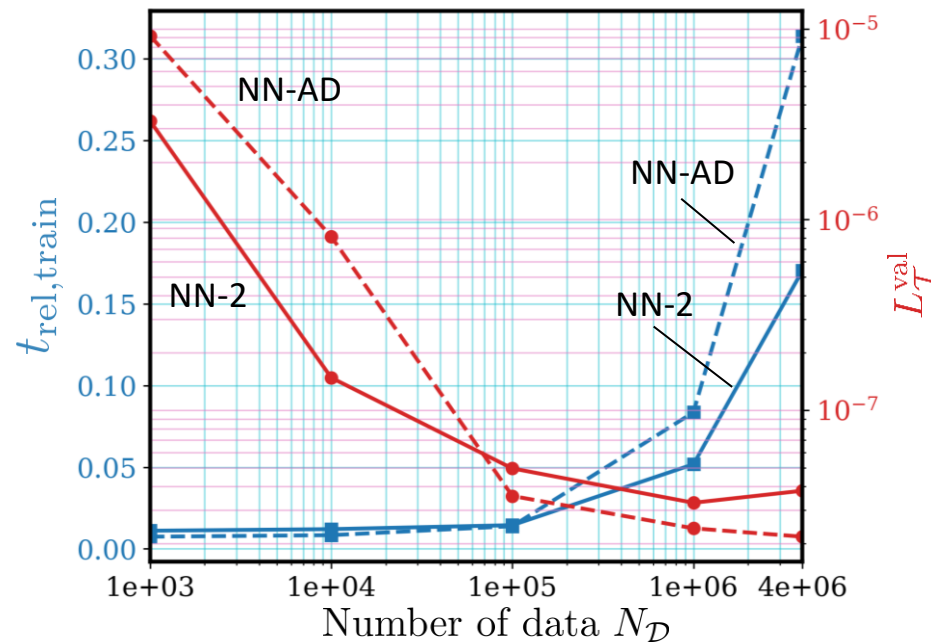
$$\mathbf{T} = K_f (\text{tr } \mathbf{E}) \mathbf{I} + 2G_f \mathbf{E}^D$$

$K_f$ $\text{N mm}^{-2}$	$G_f$ $\text{N mm}^{-2}$
$4.35 \times 10^4$	$2.99 \times 10^4$



## Cook's Membrane – Size of Dataset

- Relative training time  $t_{\text{rel,train}} = \frac{t_{\text{train}}}{t_{\text{comp,Cook}}^{\text{ref}}}$
- Validation loss  $\mathcal{L}_{\mathcal{T}}^{\text{val}} = \frac{1}{3} \sum_{i=1}^3 (T_{ij}^{\text{ref}} - T_{ij}^{\text{pred}})^2$



- Absolute percentage error  $\varepsilon = \frac{|T_{ij}^{\text{ref}} - T_{ij}^{\text{pred}}|}{|T_{ij,\text{mean}}^{\text{ref}}|} \times 100$
- Speed-up  $\text{speed-up} = \frac{t_{\text{comp}}^{\text{ref}}}{t_{\text{comp}}^{\text{DNN}}}$

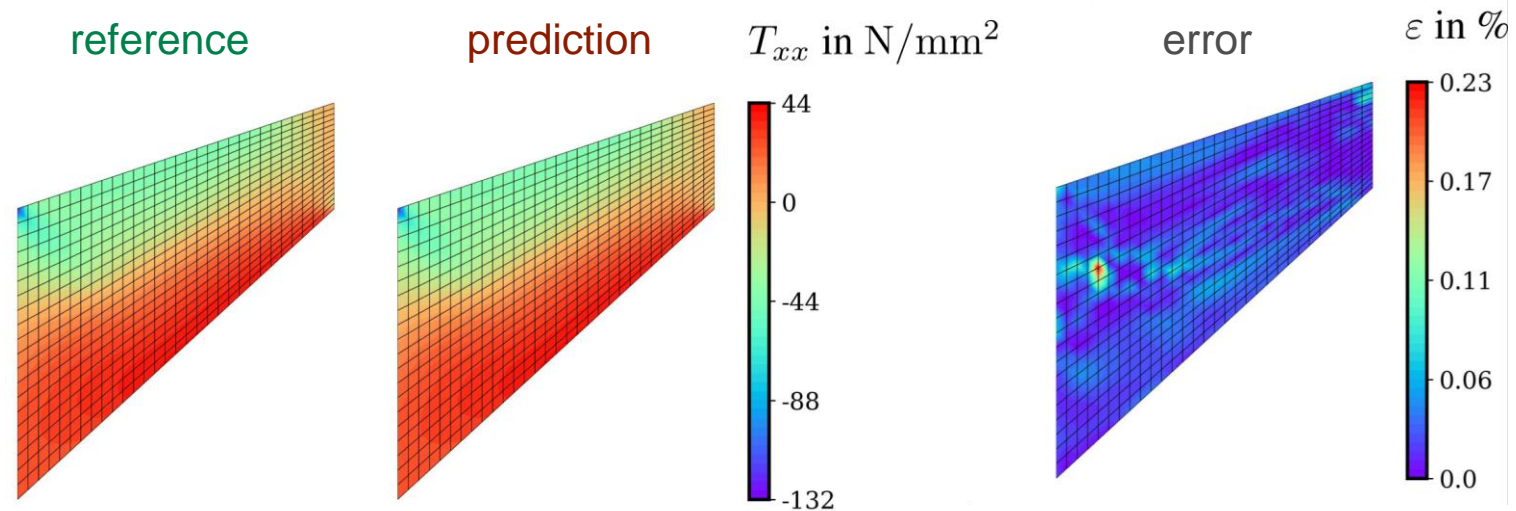
Model	$N_D$	$\varepsilon_{\text{mean}}$ (%)	$\varepsilon_{\text{std}}$ (%)	speed-up
NN-2-1	$1 \times 10^3$	0.68	0.89	242x
NN-AD-1		0.60	0.71	527x
NN-2-10	$1 \times 10^4$	0.31	0.44	292x
NN-AD-10		0.09	0.13	542x
NN-2-100	$1 \times 10^5$	0.19	0.26	287x
NN-AD-100		0.02	0.02	554x
NN-2-1000	$1 \times 10^6$	0.13	0.16	286x
NN-AD-1000		0.03	0.06	575x
NN-2-4000	$4 \times 10^6$	0.12	0.17	291x
NN-AD-4000		0.01	0.01	611x

→ Superior performance of NN-AD compared to NN-2 architecture

## Cook's Membrane – Prediction Quality

### Quality of prediction

$$\varepsilon = \frac{|T_{ij}^{\text{ref}} - T_{ij}^{\text{pred}}|}{|T_{ij,\text{mean}}^{\text{ref}}|} \times 100$$



### Improving speed-up by using Just-in-time compilation of Python code with JAX

Framework	$t_{\text{rel,train}}$	Speed-up Cook's membrane
TensorFlow	$1.39 \times 10^{-2}$	554x
JAX	$9.49 \times 10^{-3}$	5,853x

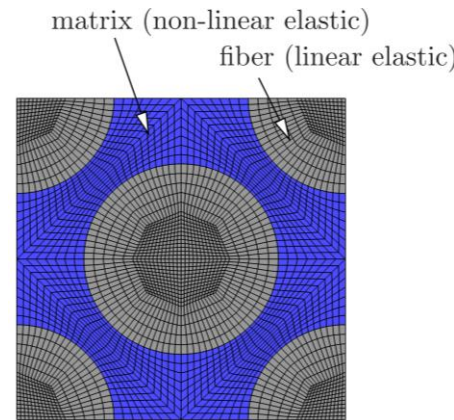
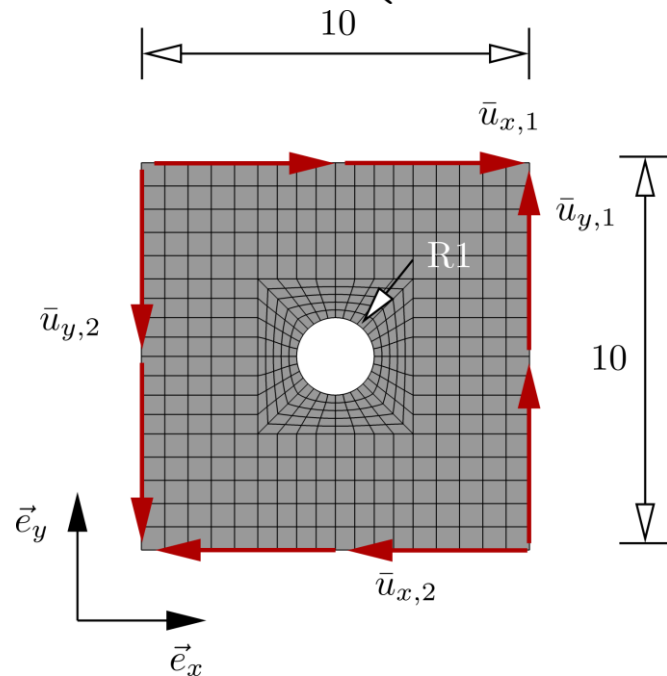
$$t_{\text{rel,train}} = \frac{t_{\text{train}}}{t_{\text{comp,Cook}}^{\text{ref}}}$$

$$\text{speed-up} = \frac{t_{\text{comp}}^{\text{ref}}}{t_{\text{comp}}^{\text{DNN}}}$$

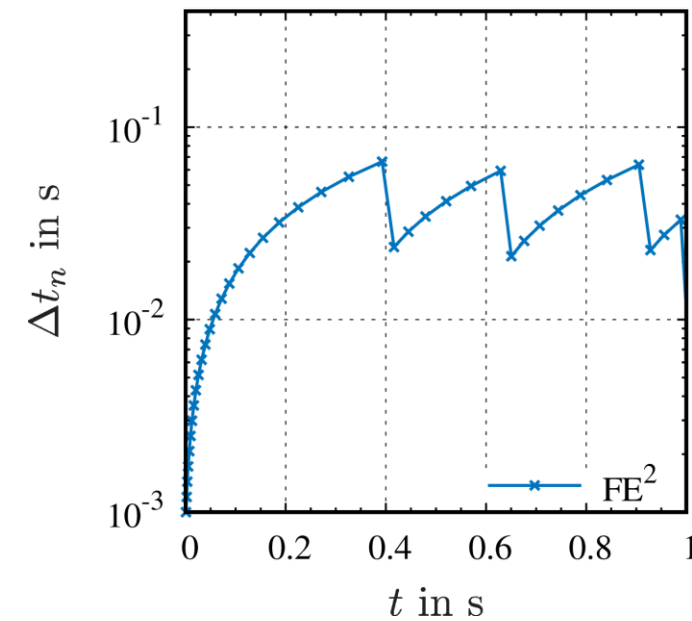
## Plate with a Hole – Step-size Control

- Load step-size control with number of Newton iterations

$$\Delta t_{\text{new}} = \Delta t_n \times \begin{cases} f_{\text{max}} = 1.2 & \text{if } N_{\text{iter}} \leq 5 \\ f_{\text{min}} = 0.3 & \text{if } N_{\text{iter}} > 15 \\ 1 & \text{if } N_{\text{iter}} > 5 \text{ and } N_{\text{iter}} \leq 15 \end{cases}$$



$$\begin{aligned} \bar{u}_{x,1}(t) &= \bar{u}_{y,1}(t) = 0,04 \text{ mm s}^{-1} t \\ \bar{u}_{x,2}(t) &= \bar{u}_{y,2}(t) = -0,04 \text{ mm s}^{-1} t \end{aligned}$$



→ DNN surrogate model can overcome load step-size limitations

## Conclusions

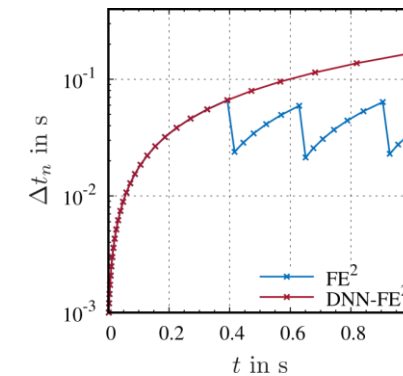
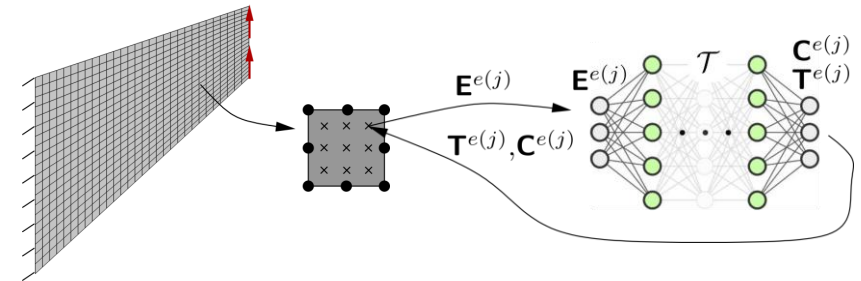
- $FE^2$  computations are computationally expensive due to numerous microscale model evaluations
- AD and Sobolev training reduce required amount of training data and increase prediction quality
- Efficient implementation using FORPy and JIT yields high speed-up ( $\sim 6,000x$ )
- Data-based DNN surrogate models can overcome load step-size limitations



Algorithm, Model  
development



Implementation





## Multilevel-Newton Algorithm vs. Newton Algorithm

$$\mathbf{G}(\mathbf{u}, \mathbf{q}) = \mathbf{0}$$

$$\mathbf{L}(\mathbf{u}, \mathbf{q}) = \mathbf{0}$$

Given: $\mathbf{u}_{n+1}^{(0)} = \mathbf{u}_n$ , $\mathbf{q}_{n+1}^{(0)} = \mathbf{q}_n$ , $\Delta t_n$ , $t_{n+1}$	
Repeat $m = 0, \dots$	
<b>local level</b> (given: $\mathbf{z} := (\mathbf{u}_{n+1}^{(m)}, \mathbf{q}_{n+1}^{(m+1)})$ ) local integration step $\mathbf{L}(\mathbf{u}_{n+1}^{(m)}, \mathbf{q}_{n+1}^{(m+1)}) = \mathbf{0} \rightarrow \mathbf{q}_{n+1}^{(m+1)}$ consistent linearization $\mathbf{z} := (\mathbf{u}_{n+1}^{(m)}, \mathbf{q}_{n+1}^{(m+1)})$ $\left[ \frac{\partial \mathbf{L}}{\partial \mathbf{q}} \right]_{\mathbf{z}} \frac{d\mathbf{q}}{d\mathbf{u}} = - \left[ \frac{\partial \mathbf{L}}{\partial \mathbf{u}} \right]_{\mathbf{z}} \rightarrow \frac{d\mathbf{q}}{d\mathbf{u}} _{\mathbf{z}}$	
<b>global level</b> solution of linear system $\left[ \left[ \frac{\partial \mathbf{G}}{\partial \mathbf{u}} \right]_{\mathbf{z}} + \left[ \frac{\partial \mathbf{G}}{\partial \mathbf{q}} \right]_{\mathbf{z}} \frac{d\mathbf{q}}{d\mathbf{u}} _{\mathbf{z}} \right] \Delta \mathbf{u}_{n+1} = -\mathbf{G}(\mathbf{z}) \rightarrow \Delta \mathbf{u}_{n+1}$ update of global variables $\mathbf{u}_{n+1}^{(m+1)} \leftarrow \mathbf{u}_{n+1}^{(m)} + \Delta \mathbf{u}_{n+1} \rightarrow \mathbf{u}_{n+1}^{(m+1)}$	
until global convergence criteria reached	

Multilevel-Newton Algorithm

Given: $\mathbf{u}_{n+1}^{(0)} = \mathbf{u}_n$ , $\mathbf{q}_{n+1}^{(0)} = \mathbf{q}_n$ , $\Delta t_n$ , $t_{n+1}$	
Repeat $m = 0, \dots$	
<b>local level</b> (given: $\mathbf{z} := (\mathbf{u}_{n+1}^{(m)}, \mathbf{q}_{n+1}^{(m)})$ ) local integration step $\left[ \frac{\partial \mathbf{L}}{\partial \mathbf{q}} \right]_{\mathbf{z}} [\mathbf{x}_1   \mathbf{x}_2] = \left[ \left[ \frac{\partial \mathbf{L}}{\partial \mathbf{u}} \right]_{\mathbf{z}}   \mathbf{L}(\mathbf{z}) \right] \rightarrow \mathbf{x}_1, \mathbf{x}_2$	
<b>global level</b> solution of linear system $\left[ \left[ \frac{\partial \mathbf{G}}{\partial \mathbf{u}} \right]_{\mathbf{z}} - \left[ \frac{\partial \mathbf{G}}{\partial \mathbf{q}} \right]_{\mathbf{z}} \mathbf{x}_1 \right] \Delta \mathbf{u}_{n+1} = -\mathbf{G}(\mathbf{z}) + \left[ \frac{\partial \mathbf{G}}{\partial \mathbf{q}} \right]_{\mathbf{z}} \mathbf{x}_2 \rightarrow \Delta \mathbf{u}_{n+1}$ update of global variables $\mathbf{u}_{n+1}^{(m+1)} \leftarrow \mathbf{u}_{n+1}^{(m)} + \Delta \mathbf{u}_{n+1} \rightarrow \mathbf{u}_{n+1}^{(m+1)}$	
<b>local level</b> computation $\Delta \mathbf{q}_{n+1} = -\mathbf{x}_1 \Delta \mathbf{u}_{n+1} - \mathbf{x}_2 \rightarrow \Delta \mathbf{q}_{n+1}$ update of local variables $\mathbf{q}_{n+1}^{(m+1)} \leftarrow \mathbf{q}_{n+1}^{(m)} + \Delta \mathbf{q}_{n+1} \rightarrow \mathbf{q}_{n+1}^{(m+1)}$	
until global convergence criteria reached	

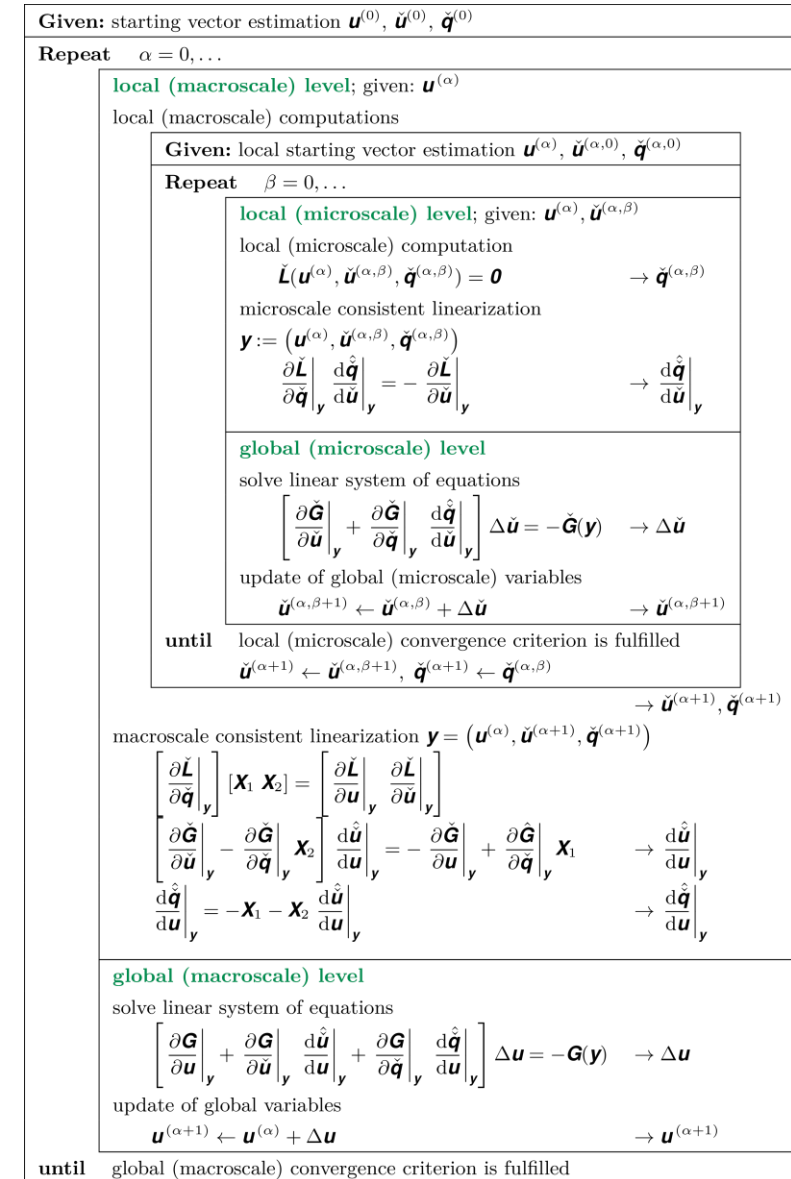
Newton Algorithm



## Multilevel-Newton Algorithm – Inelastic FE<sup>2</sup>

- Three-level Newton algorithm for inelastic problems
- Unknown quantities
  - Macroscale displacements  $\mathbf{u}$
  - Microscale displacements  $\tilde{\mathbf{u}}$
  - Microscale internal variables  $\check{\mathbf{q}}$

→ Computational costs especially for local macroscale computations



## Training Process

- Scaling of data for training efficiency  $\tilde{V}_i = \frac{\bar{V}_i - \mu_i}{\sigma_i}, \quad \tilde{C}_{ij} = \frac{\partial \tilde{T}_i}{\partial \tilde{E}_j} = \frac{\partial \tilde{T}_i}{\partial \bar{T}_i} \frac{\partial \bar{T}_i}{\partial \bar{E}_j} \frac{\partial \bar{E}_j}{\partial \tilde{E}_j} = \frac{\sigma_j}{\sigma_i} \bar{C}_{ij}$
- Weights initialized with Glorot uniform algorithm and biases with zeros initialization
- Stochastic gradient descent algorithm: Adam optimizer
- Training for 4,000 epochs with exponential decay of learning rate (decay step 1,000)

$$\eta = \eta_{\text{initial}} \gamma^{(\text{current step}/\text{decay step})} \quad \text{with} \quad \eta_{\text{initial}} = 10^{-3}, \gamma = 0.1$$

- For comparability, always 100 batches are applied
- Training/validation split 80:20
- Loss function – Mean squared error

$(\alpha, \beta)$	$\mathcal{L}_{\mathcal{T}}^{\text{val}}$	$\mathcal{L}_{\mathcal{T}'}^{\text{val}}$
(1, 0.01)	$4.84 \times 10^{-8}$	$1.35 \times 10^{-6}$
(1, 1)	$2.20 \times 10^{-8}$	$8.85 \times 10^{-8}$
(1, 100)	$3.55 \times 10^{-8}$	$2.97 \times 10^{-8}$

$$\mathcal{L} = \alpha \mathcal{L}_{\mathcal{T}} + \beta \mathcal{L}_{\mathcal{T}'} = \alpha \frac{1}{3} \sum_{i=1}^3 (T_{ij}^{\text{ref}} - T_{ij}^{\text{pred}})^2 + \beta \frac{1}{9} \sum_{i=1}^9 (C_{ij}^{\text{ref}} - C_{ij}^{\text{pred}})^2$$

## Symmetry Relations

- Since the applied constitutive models show no tension/compression-asymmetry, path-or time-dependent behavior, symmetry conditions can be applied

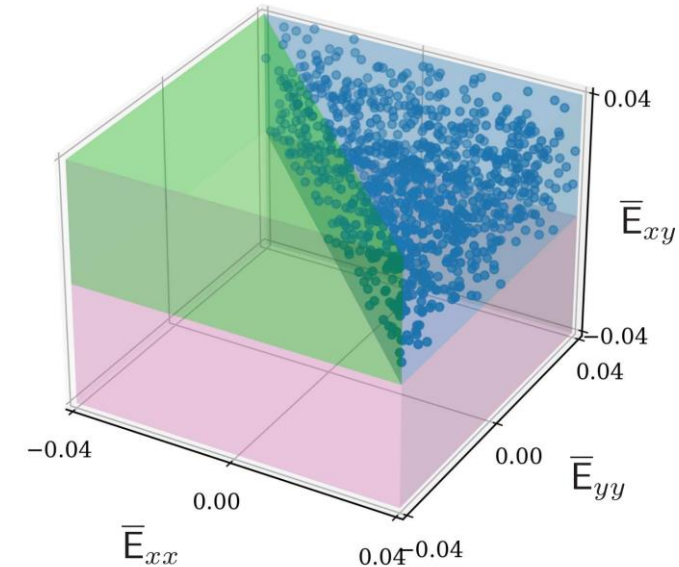
- Tension/Compression symmetry

$$\bar{T}_{xx}(-\bar{E}_{xx}, -\bar{E}_{yy}, \bar{E}_{xy}) = -\bar{T}_{xx}(\bar{E}_{xx}, \bar{E}_{yy}, \bar{E}_{xy}),$$

$$\bar{T}_{yy}(-\bar{E}_{xx}, -\bar{E}_{yy}, \bar{E}_{xy}) = -\bar{T}_{yy}(\bar{E}_{xx}, \bar{E}_{yy}, \bar{E}_{xy}).$$

- Shear symmetry

$$\bar{T}_{xy}(\bar{E}_{xx}, \bar{E}_{yy}, -\bar{E}_{xy}) = -\bar{T}_{xy}(\bar{E}_{xx}, \bar{E}_{yy}, \bar{E}_{xy}),$$



## Speed-up with Just-in-time Compilation

- Efficient implementation and fast model evaluation are crucial for surrogate model performance and speed-up
  - Just-in-time compilation (JIT) of Python code using JAX library
- Computational efficiency for TensorFlow and JAX (NN-AD-100 model)

$$t_{\text{rel,train}} = \frac{t_{\text{train}}}{t_{\text{comp,Cook}}^{\text{ref}}} \quad \text{speed-up} = \frac{t_{\text{comp}}^{\text{ref}}}{t_{\text{comp}}^{\text{DNN}}}$$

Framework	$t_{\text{rel,train}}$	Speed-up Cook's membrane	Speed-up plate with hole
TensorFlow	$1.39 \times 10^{-2}$	554x	524x
JAX	$9.49 \times 10^{-3}$	5,853x	6,048x

→ Higher speed-up and reduction in training time with JIT

## Cook's Membrane – Step-size Behavior

- Step-size behavior for NN-AD-100 (step-size control with Number of Newton iterations)

$$\Delta t_{\text{new}} = \Delta t_n \times \begin{cases} f_{\text{max}} = 1.2 & \text{if } N_{\text{iter}} \leq 5 \\ f_{\text{min}} = 0.3 & \text{if } N_{\text{iter}} > 15 \\ 1 & \text{if } N_{\text{iter}} > 5 \text{ and } N_{\text{iter}} \leq 15 \end{cases}$$

